3 Inference Rules

Logic: another thing that penguins aren’t very good at.
3 Inference Rules

- Inference Rules
- Derivation
Inference Rules: Tools for Reasoning

Syllogism (logical argument) (Aristotle, 350 BCE)
E.g.: All humans are mortal.
     Socrates is human.
     Therefore: Socrates is mortal.

First-Order Logic (Gottlob Frege, 1879 “Begriffsschrift”)
E.g.: ∀x. Human(x) \rightarrow Mortal(x)
     Human(Socrates)
     \therefore Mortal(Socrates)
Visual Notation for Inference Rules

∀x. Human(x) → Mortal(x)
Human(Socrates) 
∴ Mortal(Socrates)

Rule Schema

Premise:

Conclusion:

Variables are all-quantified

Rule Instance

Human(Socrates) → Mortal(Socrates)  Human(Socrates)
----------------------------------------
Mortal(Socrates)
Inference Rules: Overview

Structure of an inference rule:
- part of the definition of a relation (or predicate)
- has one conclusion, and zero or more premises

Meaning of an inference rule:
- expresses a reasoning step
- the conclusion holds if all premisses are true
- an argument is expressed by a derivation, which is given by a tree of rule instances (discussed later)
Predicates, Functions & Relations

Nullary Predicate $\equiv$ Set $A$

$\text{Term} = \{\text{Tru, Fls, Not Fls, Cond Fls Fls Fls Fls, ...}\}$

Unary Predicate (over $A$) $\equiv A \rightarrow \text{Bool} \equiv \text{Subset of } A$

$\text{Even} : \mathbb{N} \rightarrow \mathbb{B} = \{(0, \text{true}), (1, \text{false}), (2, \text{true}), \ldots\} \cong \{0, 2, 4, \ldots\} \subseteq \mathbb{N}$

Binary Predicate (over $A$ and $B$) $\equiv A \times B \rightarrow \text{Bool} \equiv \text{Subset of } A \times B$

$\lessdot : \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{B} = \{(0, 0), \text{false}), ((0, 1), \text{true}), \ldots, ((5, 3), \text{false}), \ldots, ((5, 7), \text{true}), \ldots\} \cong \{(0, 1), \ldots, (5, 7), \ldots\} \subseteq \mathbb{N} \times \mathbb{N}$

$t \in \text{Term} ::= \text{Tru} \mid \text{Fls} \mid \text{Not} t \mid \text{Cond} t t t$

Characteristic function

$\approx \text{SQL query over table (with schema } A, A \times B, \ldots)$
The Structure of Definitions

The definition of a concept consists of three parts:
1. **Syntax** (How to use the concept)
2. **Type** (What kind of information does it relate?)
3. **Content** (The definition itself)

**Name and syntax** (spelling, hyphenation, pronunciation)

- **se·man·tics** [səˈmantiks]
  - plural noun [usu. treated as sing.]
  - the branch of linguistics and logic concerned with meaning. ...
  - the meaning of a word, phrase, sentence, or text: such quibbling over semantics may seem petty stuff.
Function Definitions

Type

extensional meaning = semantics
{(0,1), (1,1), (2,2), (3,6), (4,24), …}

Content

intensional definition

{[(0,1), (2,2), (4,24), …]}

Prefix

Name and syntax

fac :: Int -> Int
fac 0 = 1
fac n = n * fac (n-1)

Infix

(++) :: [a] -> [a] -> [a]
[] ++ ys = ys
(x:xs) ++ ys = x:(xs ++ ys)

Inference Rules
Inference Rules

Relation Definition

Compare with function definition:

\[(++): [a] \rightarrow [a] \rightarrow [a]\]
\[[] \quad ++ \; ys = \; ys\]
\[(x:xs) \quad ++ \; ys = \; x:(xs \; ++ \; ys)\]

Judgment: Syntax & Type part of a relation definition

Content: Inference Rules: Content part of a relation definition

Name and syntax

Type

infix

\[n < n \subseteq \mathbb{N} \times \mathbb{N}\]

\[0 < 1\]

\[\ldots\]

\[k < n \quad n < m\]

\[\quad k < m\]
Inference Rules for Sets

Nullary Predicate $\equiv$ Set $A$

Term = \{Tru, Fls, Not Fls, Cond Fls Fls Fls Fls, ...\}

Inference rules provide inductive definitions of sets

- Tru $\in$ Term
- Fls $\in$ Term
- $t_1 \in$ Term
- Not $t_1 \in$ Term
- $t_2 \in$ Term
- $t_3 \in$ Term

Axiom (0 Premises)

Rule (Schema)
Representing Set Inference Rules in Idris

```idris
data Term : Type where
  Tru  : Term
  Fls  : Term
  Not : Term → Term
  Cond : Term → Term → Term → Term

syntax If [c] Then [t] Else [e] = Cond c t e
```

All but last arrows: Separating Premises

Last arrow: Separating Conclusion

Idris syntax rules can provide concrete syntax
Exercises

1. Define the sets \( \text{Stmt} \) and \( \text{Expr} \) by inference rules.

\[
\begin{align*}
  s &\in \text{Stmt} ~::=~ \text{noop} ~|~ s;s ~|~ \text{while} ~e ~\{ ~s ~\}\tag{1.1.1} \\
  e &\in \text{Expr} ~::=~ 0 ~|~ \text{succ} ~e ~|~ (e) \tag{1.1.2}
\end{align*}
\]

\[
\begin{align*}
  t &\in \text{Term} ~::=~ \text{Tru} ~|~ \text{Fls} ~|~ \text{Not} ~t ~|~ \text{Cond} ~t ~t ~t \tag{1.1.3}
\end{align*}
\]

\[
\begin{align*}
  \text{Tru} &\in \text{Term} & \quad \text{Fls} &\in \text{Term} \\
  \text{Not} &\quad t_1 \in \text{Term} \\
  \text{Cond} &\quad t_1 \in \text{Term} & \quad t_2 \in \text{Term} & \quad t_3 \in \text{Term}
\end{align*}
\]
Exercises

2. Define the syntax for Stmt and Expr using Idris data types.

\[
s \in Stmt \quad ::= \quad \text{noop} \mid s; s \mid \text{while } e \{ s \}
\]

\[
e \in Expr \quad ::= \quad 0 \mid \text{succ } e \mid (e)
\]
Inference Rules for Unary Predicates

**Unary Predicate (over A)** \(\equiv A \rightarrow \text{Bool} \equiv \text{Subset of A}\)

Even : \(\mathbb{N} \rightarrow \mathbb{B} = \{(0, \text{true}), (1, \text{false}), (2, \text{true}), \ldots\} \cong \{0, 2, 4, \ldots\} \subseteq \mathbb{N}\)

Inference rules enable the **inductive definitions** of predicates

**Inference**

\[
\frac{\text{Even}(n) \subseteq \mathbb{N}}{	ext{Even}(\text{S (S n))}} \quad \text{Judgment (syntax & type)}
\]

\(\text{Even} (0)\)

\(\text{Even} (n)\)

\(\overline{\text{Even (S (S n))}}\)
Inference Rules are Partial

\[
\begin{align*}
\text{Even}(0) \\
\text{Even}(n) \\
\text{Even}(\text{S (S n)})
\end{align*}
\]

Inference rules only state positive instances of predicates.

(in contrast to Idris functions, which should be complete)

Does \text{Even(S 0)} hold?

No, because there exists no possible derivation.

\[
\begin{align*}
even : \text{Nat} \rightarrow \text{Bool} \\
even\ 0 = \text{True} \\
even\ 1 = \text{False} \\
even\ (\text{S (S n)}) = \text{even} n
\end{align*}
\]
Representing Predicates &
Inference Rules in Idris

```
data Even : Nat → Type where
  Base : Even 0
  Step : Even n → Even (S (S n))
```

- **Type of the predicate**
- **Axiom (no premise)**
- **Rule Names**
- **Premise**
- **Conclusion**

**Premise**
```
Even(n) ⊆ ℕ
```

**Conclusion**
```
Even(0)
```

**Step**
```
Even n →
-----------
Even (S (S n))
```

A more “visual” rule representation
3. Define the unary predicate $CT(t) \subseteq \text{Term}$ that is true for all terms that contain at least one occurrence of the constant $\text{Tru}$.

$t \in \text{Term} ::= \text{Tru} \mid \text{Fls} \mid \text{Not} \; t \mid \text{Cond} \; t \; t \; t$
4. Define the unary predicate \( CT(t) \subseteq \text{Term} \) as an Idris data type.

\[
\begin{align*}
\text{data Even : Nat} \rightarrow \text{Type} \text{ where} \\
\text{Base : Even 0} \\
\text{Step : Even n} \rightarrow \text{Even (S (S n))}
\end{align*}
\]
Piazza Question

Unary predicates
Inference Rules for Binary Predicates

**Binary Predicate (over A and B)**

\[ \equiv A \times B \rightarrow \text{Bool} \equiv \text{Subset of } A \times B \]

\[ < : \mathbb{N} \times \mathbb{N} \rightarrow B = \{((0, 0), \text{false}), ((0, 1), \text{true}), \ldots, ((5, 3), \text{false}), \ldots, ((5, 7), \text{true}), \ldots\} \]

\[ \cong \{(0, 1), \ldots, (5, 7), \ldots\} \subseteq \mathbb{N} \times \mathbb{N} \]

\[ \_ < \_ \subseteq \mathbb{N} \times \mathbb{N} \]

\[ n < S n \]

Axiom schema (cf. axiom Even(0))

\[ k < n \quad n < m \]

\[ \frac{k < m}{k < m} \]
Inference Rules for Binary Predicates in Idris

\[
\text{data Less : Nat → Nat → Type where}
\]
\[
\text{Suc : Less n (S n)}
\]
\[
\text{Trans : Less k n → Less n m → Less k m}
\]

\[
\text{Suc represents, for an arbitrary n, the axiom } n < S n
\]

\[
\text{Trans combines two } < \text{ derivations into a new one}
\]

\[
\begin{align*}
\_<_&\subseteq \mathbb{N} \times \mathbb{N} \\
\frac{n < S n \quad k < n \quad n < m}{k < m}
\end{align*}
\]

\[
\text{A more “visual” rule representation}
\]

\[
\text{Trans : Less k n → Less n m →}
\]
\[
\text{Less k m}
\]
Examples

Name for the fact

A particular relationship

Value representing the fact

\[ \text{less01} : \text{Less} \ 0 \ 1 \]
\[ \text{less01} = \text{Suc} \]

\[ \text{less02} : \text{Less} \ 0 \ 2 \]
\[ \text{less02} = \text{Trans Suc Suc} \]

\[ \text{Suc} \rightarrow \text{Less 0 1} \]
\[ \text{Suc} \rightarrow \text{Less 1 2} \]
\[ \text{Trans Suc Suc} \rightarrow \text{Less 0 2} \]
Piazza Question

The Less predicate
Inference Rules for Binary Predicates in Idris

Equality is a binary relation

\[
data (\_\_ : a \to b \to \text{Type}) \text{ where}
\]

\[
\text{Refl} : x = x
\]

Every element of a type is equal to itself (reflexivity)

\[
\forall A : \_\_ = \_ \subseteq A \times A
\]

\[
x = x
\]

A single axiom only

\[\text{Refl} \]
represents, for an arbitrary \( x \), the axiom \( x = x \)
Curry-Howard Isomorphism

\[ \text{program} : \text{Type} \]

\[ \text{Proof for Proposition} \]
Constructors as Proofs (1)

```
data Term : Type where
    Tru  : Term
    Fls  : Term
    Not  : Term → Term
    Cond : Term → Term → Term → Term
```

*Tru* is a value of type *Term* proof that the proposition *Term* is true

*Not* is a function of type *Term → Term* rule that, given a proof for *Term*, yields another proof for *Term*

"There exist some terms" (Boring!)
Constructors as Proofs (2)

```haskell
data Even : Nat → Type where
  Base : Even 0
  Step : Even n → Even (S (S n))
```

- **Base** is a value of type `Even 0` — "proof" that the proposition `Even 0` is true.
- **Step** is a function of type `Even n → Even (S (S n))` — rule that, given a proof for `Even n`, yields a proof for `Even (S (S n))`.

Axiom, i.e. 0 is even by definition.
Piazza Question

The Even predicate
Constructors as Proofs (3)

Data definition for less than relation:

\[
\text{data Less : Nat → Nat → Type where}
\]

\[
\begin{align*}
\text{Suc} & : \text{Less } n \ (S \ n) \\
\text{Trans} & : \text{Less } k \ n \ → \ \text{Less } n \ m \ → \ \text{Less } k \ m
\end{align*}
\]
More Interesting Types

**Inference Rules**

- **sort**: \( \text{List} \rightarrow \text{SortedList} \)
- **\([\_]\)**: \( \text{Term} \rightarrow \text{Value} \)

\[ (\text{Small-step semantics}) \]

- Term \( t \) reduces in one step to \( t_1 \)

\[ (\text{Big-step semantics}) \]

- Term \( t \) reduces in many steps to \( [t] \)

\[ (\text{Denotational semantics}) \]

- Value denoted by \( t \)
Translating Inference Rules into Idris Data Types

(1) Define syntax and type of predicate in the `data ... where` header
(2) For each rule, define a constructor (the “name” of the rule)
(3) Write the conclusion of a rule as the result type of the constructor
(4) Write the premises of a rule as the argument types of the constructor

\[ \_ \_ \_ \leq \_ \_ \subseteq \mathbb{N} \times \mathbb{N} \]

\[ n < S \_ \_ \_ \]

\[ k < n \hspace{1em} n < m \]

\[ \frac{k < n \hspace{1em} n < m}{k < m} \]

\[ \text{data Less} : \text{Nat} \to \text{Nat} \to \text{Type} \text{ where} \]
\[ \text{Suc} : \text{Less } n \ (S \ n) \]
\[ \text{Trans} : \text{Less } k \ n \ \to \ \text{Less } n \ m \ \to \]
\[ \hspace{1em} \text{Less } k \ m \]
Syntax: A Special Case

(1) Define syntax and type of predicate in the \texttt{data} \ldots \texttt{where} header
(2) For each \textit{rule}, define a \textit{constructor} (the “name” of the rule)
(3) Write the \textit{conclusion} of a rule as the \textit{result type} of the constructor
(4) Write the \textit{premises} of a rule as the \textit{argument types} of the constructor

\begin{align*}
\text{Term is a set} & \quad \text{Tru} \in \text{Term} \\
\text{Fls} \in \text{Term} & \\
\frac{t_1 \in \text{Term}}{\text{Not} \ t_1 \in \text{Term}}
\end{align*}

\texttt{data Term : Type where}
\begin{align*}
\text{Tru} : & \text{Term} \\
\text{Fls} : & \text{Term} \\
\text{Not} : & \text{Term} \rightarrow \text{Term} \\
\text{Cond} : & \text{Term} \rightarrow \text{Term} \rightarrow \text{Term} \rightarrow \text{Term} \rightarrow \text{Term}
\end{align*}

\begin{align*}
\frac{t_1 \in \text{Term} \quad t_2 \in \text{Term} \quad t_3 \in \text{Term}}{\text{Cond} \ t_1 \ t_2 \ t_3 \in \text{Term}}
\end{align*}
3 Inference Rules

- Inference Rules
- Derivation
Derivation = Proof Tree

Is Cond Fls Tru (Not Fls) ∈ Term?

Rule System

\[
\begin{align*}
\text{Tru} & \in \text{Term} \\
\text{Fls} & \in \text{Term} \\
\end{align*}
\]

\[
\begin{align*}
\text{Cond} & \; t_1 \; t_2 \; t_3 \in \text{Term} \\
\text{Not} & \; t_1 \in \text{Term} \\
\end{align*}
\]

Instance

\[
\begin{align*}
\text{Fls} & \in \text{Term} \\
\text{Tru} & \in \text{Term} \\
\text{Not Fls} & \in \text{Term} \\
\text{Cond Fls Tru (Not Fls)} & \in \text{Term} \\
\end{align*}
\]

Instance

\[
t_1 = \text{Fls}, t_2 = \text{Tru}, t_3 = \text{Not Fls}
\]

\[
t_1 = \text{Fls}
\]
Derivation Tree

Nodes: judgments, Leaves: Axioms
Edges: link premises to conclusions

Inference Rule: Template for constructing derivation trees

Rule (Instance of a Rule Schema)
Instantiating Rules Using Names

Rule Name

If

Tru

\[ \text{Tru} \in \text{Term} \]

Fls

\[ \text{Fls} \in \text{Term} \]

\[ t_1 \in \text{Term} \]

\[ t_2 \in \text{Term} \]

\[ t_3 \in \text{Term} \]

Cond

\[ t_1 \in \text{Term} \]

\[ t_2 \in \text{Term} \]

\[ t_3 \in \text{Term} \]

Not

\[ \text{Not} \in \text{Term} \]

\[ t_1 \in \text{Term} \]

Which rule was used?

If

Fls

\[ \text{Fls} \in \text{Term} \]

Tru

\[ \text{Tru} \in \text{Term} \]

\[ \text{Fls} \in \text{Term} \]

\[ \text{Tru} \in \text{Term} \]

Cond

\[ \text{Fls} \in \text{Term} \]

\[ \text{Tru} \in \text{Term} \]

\[ \text{Fls} \in \text{Term} \]

\[ \text{Not} \in \text{Term} \]

\[ \text{Fls} \in \text{Term} \]

\[ \text{Not} \in \text{Term} \]

\[ \text{Fls} \in \text{Term} \]
Inference Rules

Derivation in Idris

Is $\text{Cond Fls Tru (Not Fls)} \in \text{Term}$?

Yes, because we can construct a type-correct expression out of the proof constructors
Exercises

5. Draw the derivation tree for the following judgment.

\[ \text{while succ 0 \{noop; while 0 \{noop\}\} ∈ Stmt} \]
Exercises

6. Express the following judgment in Idris.
   \[
   \text{while succ 0 \{noop; while 0 \{noop\}\} } \in \text{Stmt}
   \]

7. Create a derivation for the judgment in Idris.

---

```
data Expr : Type where
  Zero : Expr
  Succ : Expr \rightarrow Expr

data Stmt : Type where
  Noop : Stmt
  Seq : Stmt \rightarrow Stmt \rightarrow Stmt
  While : Expr \rightarrow Stmt \rightarrow Stmt
```
Exercises

8. Draw a derivation tree and write a corresponding Idris expression (= proof) for the judgment Even(2).

```idris
data Even : Nat → Type where
  Base : Even 0
  Step : Even n → Even (S (S n))
```

\[
\begin{align*}
\text{Even}(0) \\
\text{Even}(n) \\
\text{Even}(S \ (S \ n))
\end{align*}
\]
Exercises

9. Draw a derivation tree and write a corresponding Idris expression (= proof) for the judgment $0 < 3$.

\[
\frac{k < n \quad n < m}{k < m}
\]

data Less : Nat \to Nat \to Type where
Suc : Less n (S n)
Trans : Less k n \to Less n m \to Less k m
Exercises

10. Explain all occurrences of the symbols 0, S, and Suc

less01 : Less 0 (S 0)
less01 = Suc

data Less : Nat → Nat → Type where
  Suc   : Less n (S n)
  Trans : Less k n → Less n m → Less k m

11. Which of the following definitions are correct proofs of the lemma \( \text{Less } 0 \ 2 \)?

lemma : Less 0 2
lemma = Trans Suc Suc

lemma = Trans less01 Suc
lemma = Trans Suc less01
lemma = Trans (Suc \{0\}) Suc
lemma = Trans less01 (Suc \{0\})
lemma = Trans less01 less01
lemma = Trans less01 (Suc \{1\})